6. An airline wants to manage boarding passengers using a **Priority Queue implemented with a Heap**.

Class	Priority
First Class	3
Business Class	2
Economy Class	1

```
public class Passenger {
  private String name;
  private int priority;

  public Passenger(String name, int priority) {
    this.name = name;
    this.priority = priority;
  }

  public String getName() {
    return name;
  }

  public int getPriority() {
    return priority;
  }

  public String toString() {
    return name + " (Priority " + priority + ")";
  }
}
```

Write the HeapQueue Class using Array.

```
public static final int FIRST_CLASS = 3;
public static final int BUSINESS = 2;
public static final int ECONOMY = 1;


public class HeapQueue {
   private Passenger[] heap;
   private int size;
   public HeapQueue() {
      heap = new Passenger[10]; // Initial capacity
      size = 0;
}
```

```
// Add a passenger to the heap
public void add(Passenger passenger) {
    if (size == heap.length) {
         UI.println("Heap is full");
          return;
    }
    heap[size]=passenger;

    UI.println(passenger.getName() + " is added!");
    size++;

    heapifyUp(size - 1);
}


// Remove and return highest priority passenger
public Passenger poll() {

   if (isEmpty()) {
      UI.println("Heap is empty");
      return null;
   }

   Passenger max = heap[0];
   heap[0] = heap[size - 1];
   size--;
   heapifyDown(0);

   return max;
}
```

```
// Return highest priority passenger without removing


public Passenger peek() {

   if (isEmpty()) {
      UI.println("Heap is empty");
      return null;
  }

  return heap[0];
}

private void heapifyUp(int index) {

   while (index>0) {
      int parentIndex = (index-1)/2;

      if (heap[index].getPriority()>heap[parentIndex].getPriority()) {
         Passenger temp =heap[index];
         heap[index]= heap[parentIndex];
         heap[parentIndex] = temp;

         index = parentIndex;
      } else{
        break;
      }
   }
}
```

```
private void heapifyDown(int index) {
   while (true) {
      int largest = index;
      int leftChildIndex = 2 * index + 1;
      int rightChildIndex = 2 * index + 2;

      if (leftChildIndex<size &&
heap[leftChildIndex].getPriority(>heap[largest].getPriority())

      {
         largest=leftChildIndex;
      }

      if (rightChildIndex<size &&
      heap[rightChildIndex].getPriority()
      >heap[largest].getPriority()) {

         largest = rightChildIndex;
      }

      if (largest != index) {
         Passenger temp = heap[index];
         heap[index] = heap[largest];
         heap[largest] = temp;

         index = largest;
      } else {
      break;
      }
   }
}

public boolean isEmpty() {
   return size == 0;
  }
}
```